SObjectizer: a C++ tool for Concurrent Programming

with Actors, Pub/Sub and CSP-channels

Italian C++ Community Meetup

Nicolai Grodzitski

Introduction

Multithreading is used for...

...Parallel* and Concurrent** Computing.

Those are very different approaches used for solving very different tasks:

- parallel computing: video encoding
- concurrent computing: database server

SObjectizer is for Concurrent Computing only.

* https://en.wikipedia.org/wiki/Parallel_computing
** https://en.wikipedia.org/wiki/Concurrent_computing

Shared mutable state is the root of all evil

The dealing with shared mutable state is hard and error-prone.

SObjectizer allows to avoid shared mutable state at all.

Let's make an app from separate entities.
Each entity owns its private mutable state.

All interactions between entities only via async messages.

Well known approaches

Actor Model*

Communicating Sequential Processes**

SObjectizer allows to use elements of those models in your apps.

* https://en.wikipedia.org/wiki/Actor_model
** https://en.wikipedia.org/wiki/Communicating_sequential_processes

A bit of history

There was SCADA Objectizer

A project in Development Bureau of System Programming in Homyel, Belarus (1996-2000).

The goal: object-oriented SCADA system.

The main idea of agents with states and interactions via async messages.

The SCADA Objectizer died in 2000.

SObjectizer-4

A new project SObjectizer-4. Intervale JSC, Moscow, 2002.

Agents with states and async messages were borrowed from predecessor.

SObjectizer-4 was open-sourced* in 2006.
Its evolution stopped soon after that.

* https://sourceforge.net/projects/sobjectizer/

SObjectizer-5

The development started in 2010.

We took SObjectizer-4 and rebuilt it completely.

And then add many new features.

SObjectizer was created for production

SObjectizer is not an experimental project.
And has never been.

SObjectizer was used in business-critical projects from the very beginning.

Breaking changes in SObjectizer are rare and we approach to them very carefully.

For example:

- branch 5.5 evolved without big compatibility breaks more than 5 years;
- there was just one compatibility break in the last two years.

SObjectizer-5 is an OpenSource project

SObjectizer-5 was open-sourced in 2013 on SourceForge*.

SourceForge still contains a lot of docs for older versions of SObjectizer-5.

Now SObjectizer-5 lives on GitHub**, including the docs***.

* https://sourceforge.net/projects/sobjectizer/
** https://github.com/Stiffstream/sobjectizer
*** https://github.com/Stiffstream/sobjectizer/wiki

Where SObjectizer was used?

- SMS/USSD traffic service;
- financial transaction handling;
- electronic trading;
- software parameters monitoring;
- automatic control of the theatre's scenery*;
- machine learning;
- prototyping of distributed data-acquisition software;
- components of DMP in an advertising platform;
- components of an online game;
- multitheading socks5/http1.1 proxy-server.

* https://habr.com/en/post/452464/

A brief introduction to SObjectizer-5.7

Two main styles

Agents, mboxes, dispatchers...

Raw threads + mchains.

Both styles can be mixed inside one app.

Agents, mboxes, dispatchers

Let's speak about the main style of SObjectizer-based programming...

The ingredients

There are:

- **agents**, coops of agents;
- **messages**, subscriptions;
- mboxes (message boxes);
- dispatchers.

Messages

All interactions between agents via async messages only.

Messages are objects of user types.

Message type is a key for searching of a message handler.

An example of a message

```
struct request : public so_5::message_t
{
  std::int64_t id_;
  std::map<std::string, std::string> params_;
  std::vector<std::uint8_t> payload_;
  std::chrono::steady_clock::timepoint deadline_;

  request(
        std::int64_t id,
        std::map<std::string, std::string> params,
        std::vector<std::uint8_t> payload,
        std::chrono::steady_clock::timeout deadline)
    :  id_(id), params_(std::move(params)), payload_(std::move(payload)), deadline_(deadline)
  {}
};
```

No inheritance from so_5::message_t

```
struct request
{
  std::int64_t id_;
  std::map<std::string, std::string> params_;
  std::vector<std::uint8_t> payload_;
  std::chrono::steady_clock::timepoint deadline_;
};
```

There are also signals

```
struct get_status : public so_5::signal_t {};
```

Message sending

```
// Send immediately.
so_5::send<request>(target,
    // Will be forwarded to the constructor of `request`.
    make_id(), make_params(), make_payload(), calculate_deadline());

// Send a signal immediately.
so_5::send<get_status>(target);
```

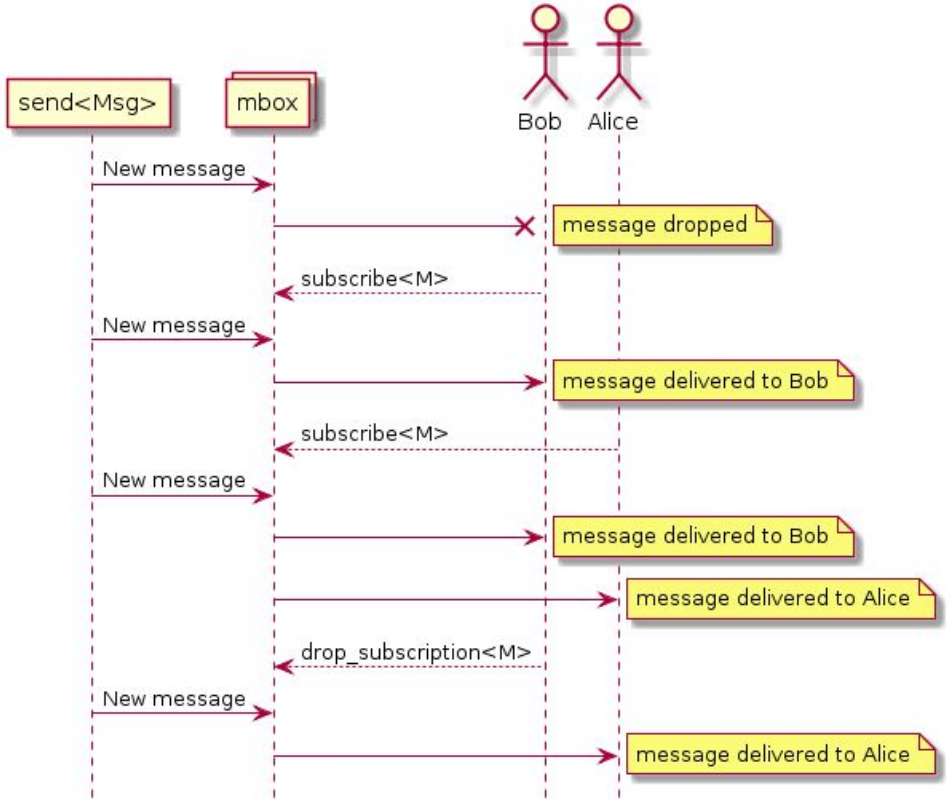
What is the Target for a send()?

In the classical Actor Model a message is sent to an actor.

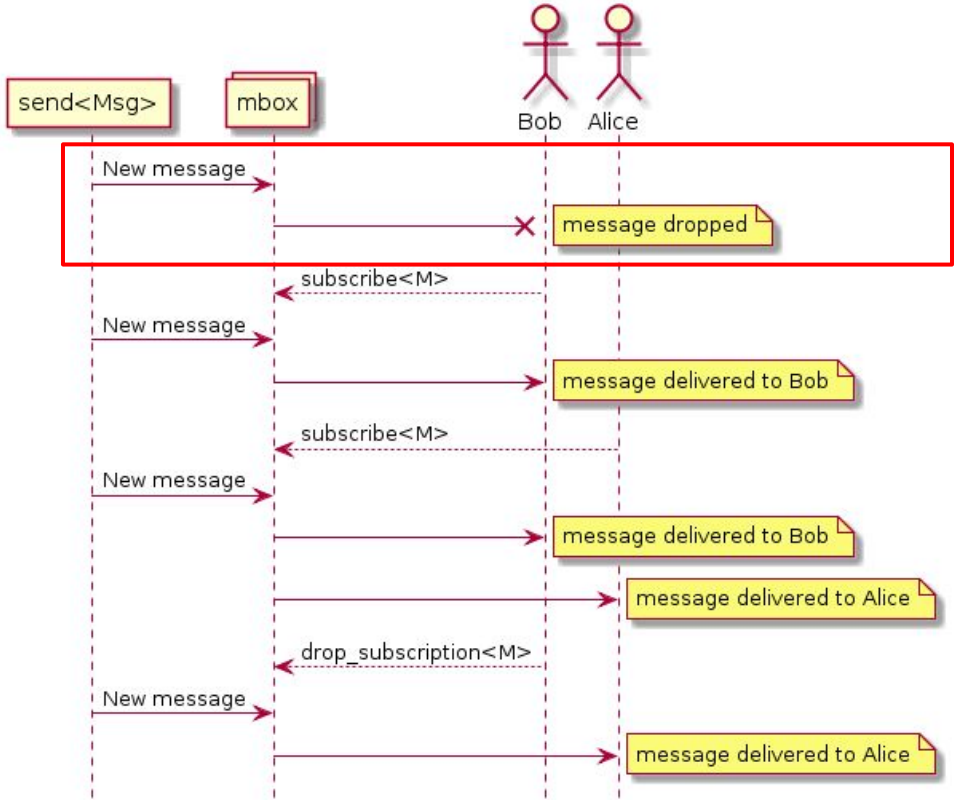
But in SObjectizer all messages are going to mboxes (message boxes).

An agent has to subscribe to a message from a mbox...

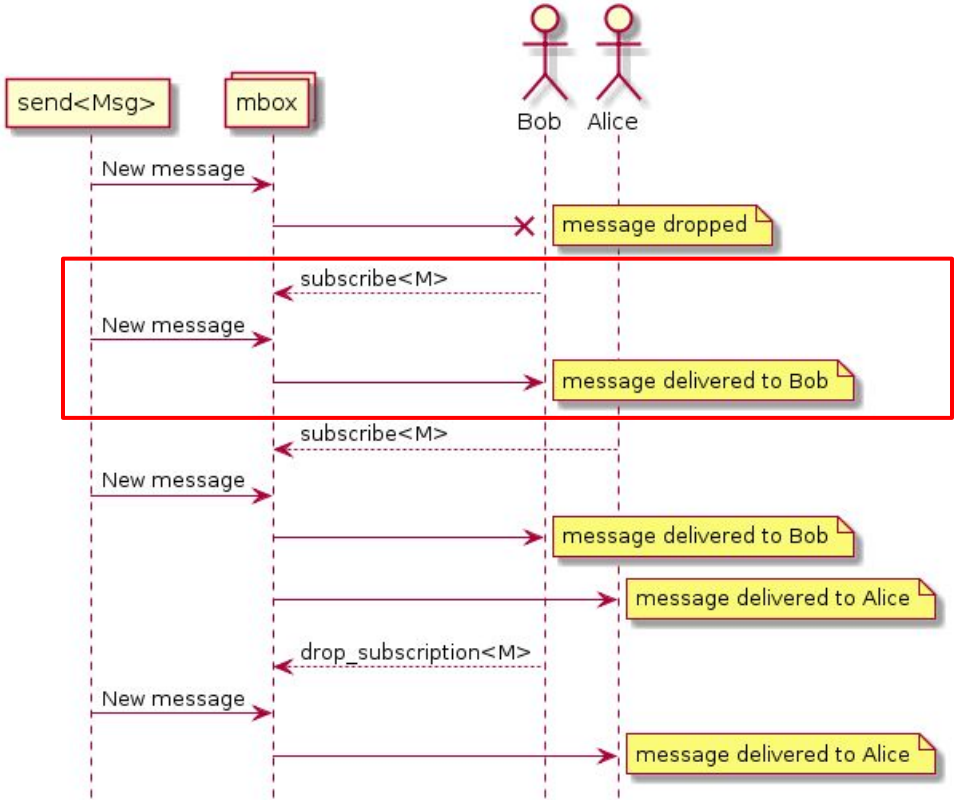
Mbox and subscription



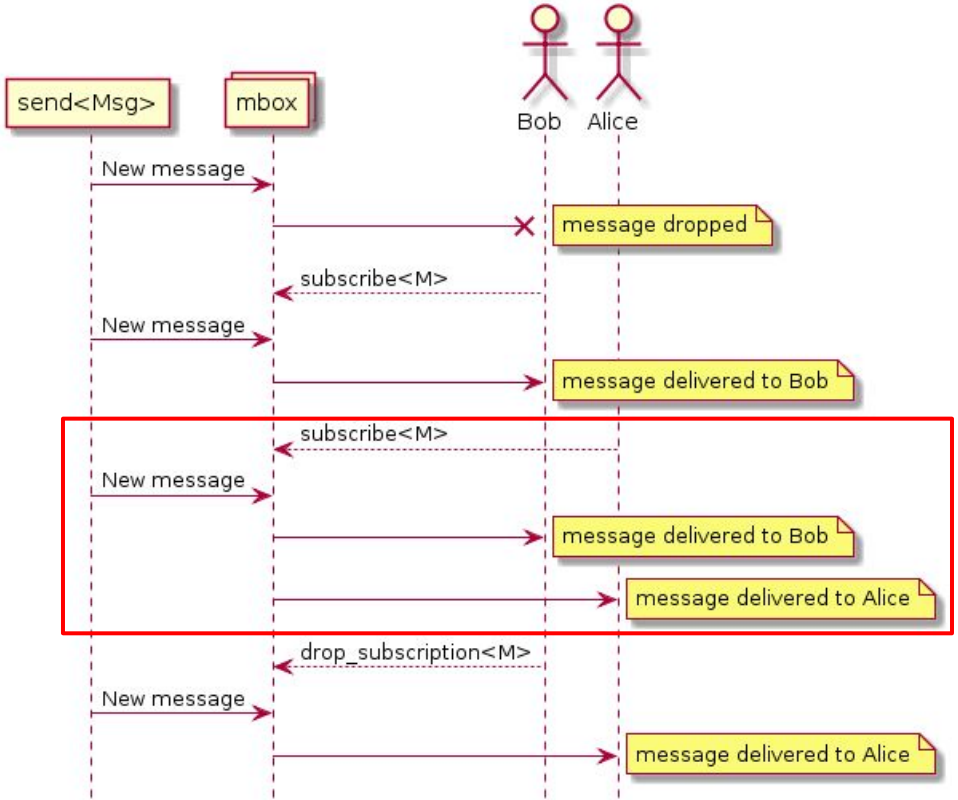
Mbox and subscription



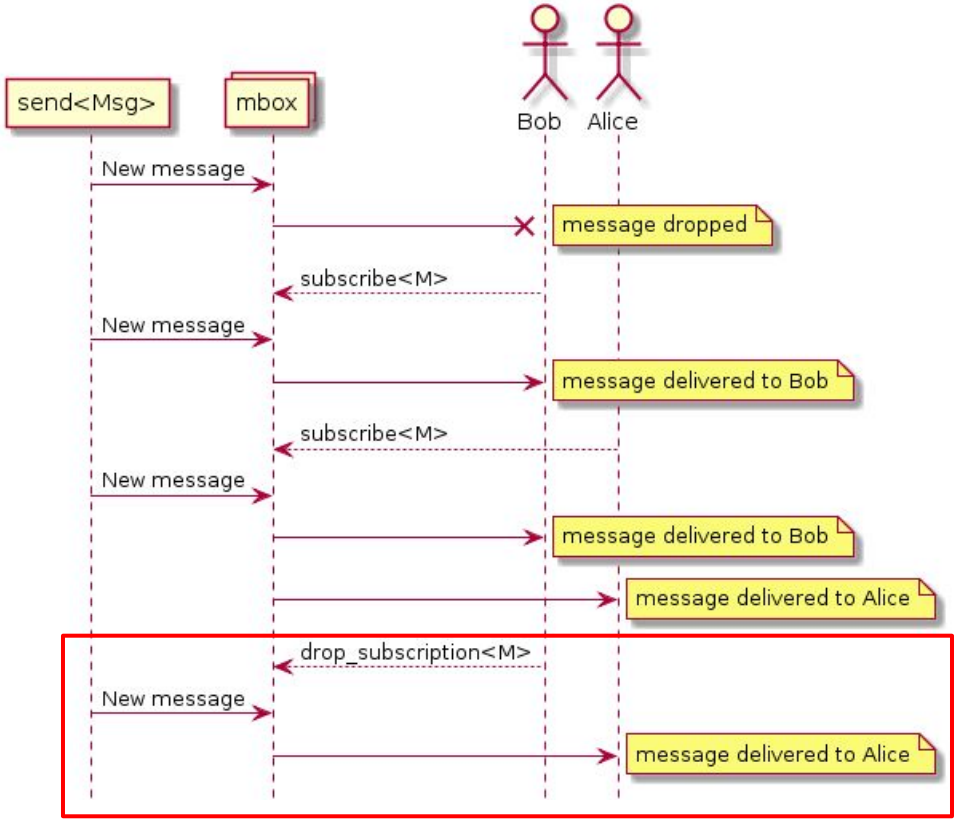
Mbox and subscription



Mbox and subscription



Mbox and subscription



Two types of mboxes

MPMC (Multi-Producer/Multi-Consumer)

- anyone can send;
- anyone can subscribe;
- all subscribers receive a message (broadcasting).

MPSC (Multi-Producer/Single-Consumer, aka direct_mbox)

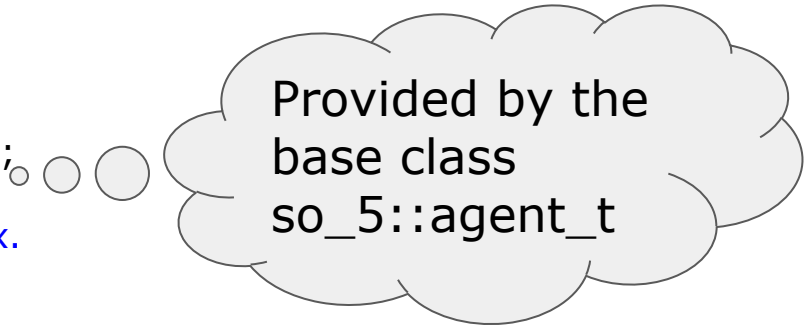
- anyone can send;
- only owner can subscribe.

Subscription

```
// Subscription to a message from a mbox.
so_subscribe(some_mbox).event(... /* handler */);

// Subscription to a message from own MPSC mbox.
so_subscribe_self().event(... /* handler */);
```

Subscription

```
// Subscription to a message from a mbox.
so_subscribe(some_mbox).event(... /* handler */);

// Subscription to a message from own MPSC mbox.
so_subscribe_self().event(... /* handler */);
```



Dispatchers

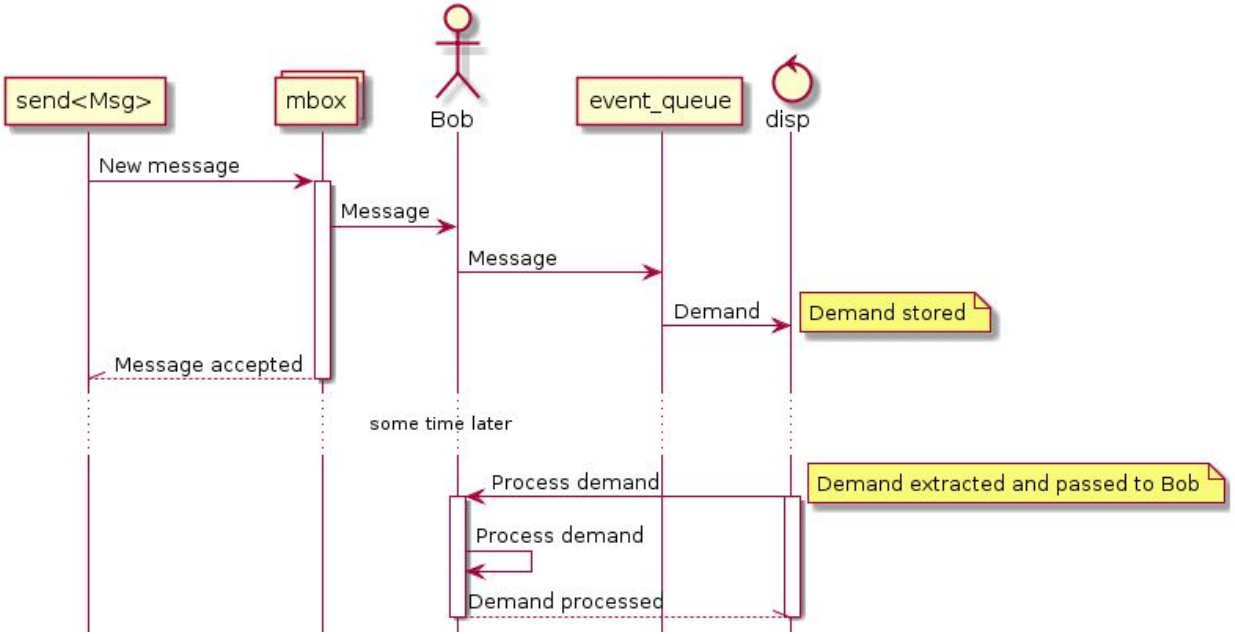
Dispatchers provide worker threads for agents.

A dispatcher selects when and where agent's event handler will be called.

There could be many dispatchers inside an app.

Every agent should be bound to some dispatcher.

Dispatchers



There are several standard dispatchers

one_thread
active_group
active_obj
thread_pool
adv_thread_pool
prio_dedicated_threads::one_per_prio
prio_one_thread::quoted_round_robin
prio_one_thread::strictly_ordered

There are several standard dispatchers

one_thread
active_group
active_obj
thread_pool
adv_thread_pool
prio_dedicated_threads::one_per_prio
prio_one_thread::quoted_round_robin
prio_one_thread::strictly_ordered

There are several standard dispatchers

one_thread
active_group
active_obj
thread_pool
adv_thread_pool
prio_dedicated_threads::one_per_prio
prio_one_thread::quoted_round_robin
prio_one_thread::strictly_ordered

There are several standard dispatchers

one_thread
active_group
active_obj
thread_pool
adv_thread_pool
prio_dedicated_threads::one_per_prio
prio_one_thread::quoted_round_robin
prio_one_thread::strictly_ordered

There are several standard dispatchers

one_thread
active_group
active_obj
thread_pool
adv_thread_pool
prio_dedicated_threads::one_per_prio
prio_one_thread::quoted_round_robin
prio_one_thread::strictly_ordered

There are several standard dispatchers

one_thread
active_group
active_obj
thread_pool
adv_thread_pool
prio_dedicated_threads::one_per_prio
prio_one_thread::quoted_round_robin
prio_one_thread::strictly_ordered

Agents

Objects of classes inherited from **so_5::agent_t**:

```
class hello_world final : public so_5::agent_t {
public :
  using so_5::agent_t::agent_t;

  void so_evt_start() override {
    std::cout << "Hello World!" << std::endl;
    so_deregister_agent_coop_normally();
  }
};
```

Agents

```
class hello_world final : public so_5::agent_t {
public :
  using so_5::agent_t::agent_t;

  void so_define_agent() override {
    so_subscribe_self().event(
      &hello_world::hello );
  }

  void so_evt_start() override {
    std::cout << "Starting agent..." << std::endl;
    so_5::send<msg_hello>(so_direct_mbox);
  }
// ...
// …
void so_evt_finish() override {
  std::cout << "Finish agent..." << std::endl;
}

  void hello_world::hello_delay( const msg_hello &  ) {
    std::cout << "Hello World!"<<std::endl;
  }
};
```

Agents

```
class hello_world final : public so_5::agent_t {
public :
  using so_5::agent_t::agent_t;

  void so_define_agent() override {
    so_subscribe_self().event(
      &hello_world::hello );
  }

  void so_evt_start() override {
    std::cout << "Starting agent..." << std::endl;
    so_5::send<msg_hello>(so_direct_mbox);
  }
// ...
// …
void so_evt_finish() override {
  std::cout << "Finish agent..." << std::endl;
}

  void hello_world::hello_delay( const msg_hello &  ) {
    std::cout << "Hello World!"<<std::endl;
  }
};
```

Agents

```
class hello_world final : public so_5::agent_t {
public :
  using so_5::agent_t::agent_t;

  void so_define_agent() override {
    so_subscribe_self().event(
      &hello_world::hello );
  }

  void so_evt_start() override {
    std::cout << "Starting agent..." << std::endl;
    so_5::send<msg_hello>(so_direct_mbox);
  }
// ...
```

```
// …
void so_evt_finish() override {
  std::cout << "Finish agent..." << std::endl;
}

void hello_world::hello_delay( const msg_hello &  ) {
  std::cout << "Hello World!"<<std::endl;
}
};
```

Agents

```
class hello_world final : public so_5::agent_t {
public :
  using so_5::agent_t::agent_t;

  void so_define_agent() override {
    so_subscribe_self().event(
      &hello_world::hello );
  }

  void so_evt_start() override {
    std::cout << "Starting agent..." << std::endl;
    so_5::send<msg_hello>(so_direct_mbox);
  }
// ...
}
```

```
// Message definition:
struct msg_hello { };
```

```
void hello_world::hello_delay( const msg_hello & ) {
  std::cout << "Hello World!"<<std::endl;
}
};
```

Agents

```
class hello_world final : public so_5::agent_t {
public :
  using so_5::agent_t::agent_t;

  void so_define_agent() override {
   so_subscribe_self().event(
     &hello_world::hello );
  }

  void so_evt_start() override {
    std::cout << "Starting agent..." << std::endl;
    so_5::send<msg_hello>(so_direct_mbox);
  }
// ...
```

```
// ...
void so_evt_finish() override {
  std::cout << "Finish agent..." << std::endl;
}

  void hello_world::hello_delay( const msg_hello &  ) {
    std::cout << "Hello World!"<<std::endl;
  }
};
```

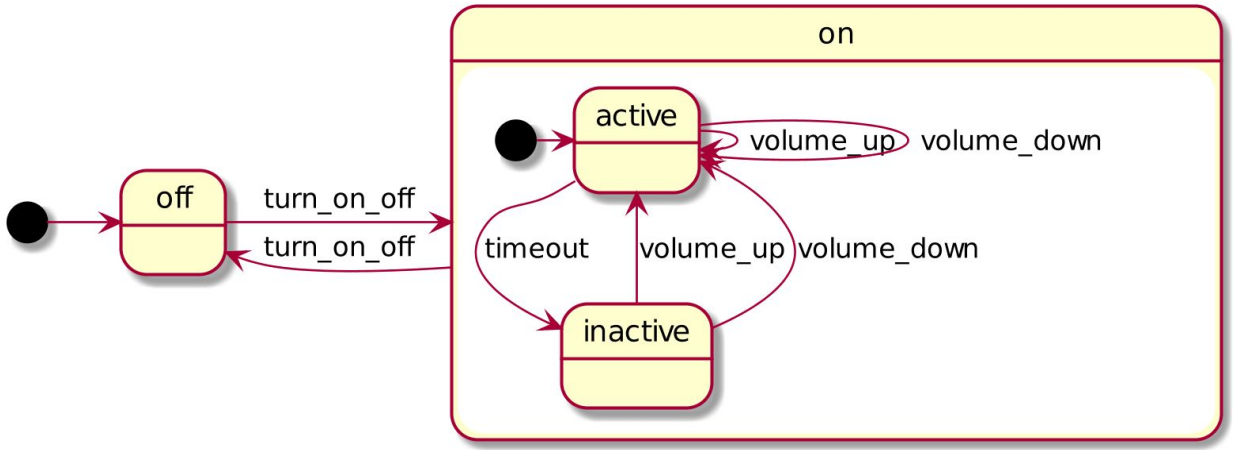
There are agent's states

Agent's states allow to implement complex hierarchical finite-state machines.

Supported HFSM features:

- nested states;
- on_enter/on_exit handlers;
- time_limit for a state;
- deep/shallow history.

An example of an agent with states



An example of an agent with states

```
class player_demo final : public so_5::agent_t {
  state_t off{this, "off"},
        on{this, "on"},
        active{initial_substate_of{on}, "active"},
        inactive{substate_of{on}, "inactive"};
  ...
```

An example of an agent with states

```
class player_demo final : public so_5::agent_t {
  state_t off{this, "off"},
        on{this, "on"},
        active{initial_substate_of{on}, "active"},
        inactive{substate_of{on}, "inactive"};
  ...
```

An example of an agent with states

```
class player_demo final : public so_5::agent_t {
  state_t off{this, "off"},
        on{this, "on"},
        active{initial_substate_of{on}, "active"},
        inactive{substate_of{on}, "inactive"};
  ...
```

An example of an agent with states

```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

An example of an agent with states

```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

An example of an agent with states

```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

An example of an agent with states

```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

An example of an agent with states

```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

An example of an agent with states

```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

An example of an agent with states

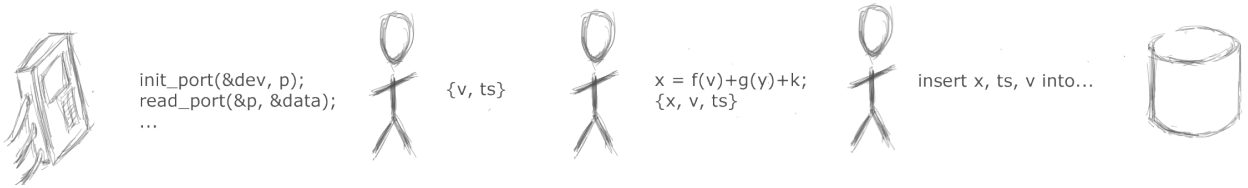
```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

An example of an agent with states

```
void so_define_agent() override {
    this >>= off;

    off.event([&](mhood_t<turn_on_off>){... /* turn the device on */});

    on.event([&](mhood_t<turn_on_off>{... /* turn the device off */});

    active
        .on_enter([&]{... /* turn the screen on */})
        .on_exit([&]{... /* turn the screen off */})
        .time_limit(15s, inactive)
        .event([&](mhood_t<volume_up> cmd){... /* increase the volume */})
        .event([&](mhood_t<volume_down> cmd){... /* decrease the volume */});

    inactive
        .transfer_to_state<volume_up>(active)
        .transfer_to_state<volume_down>(active);
}
```

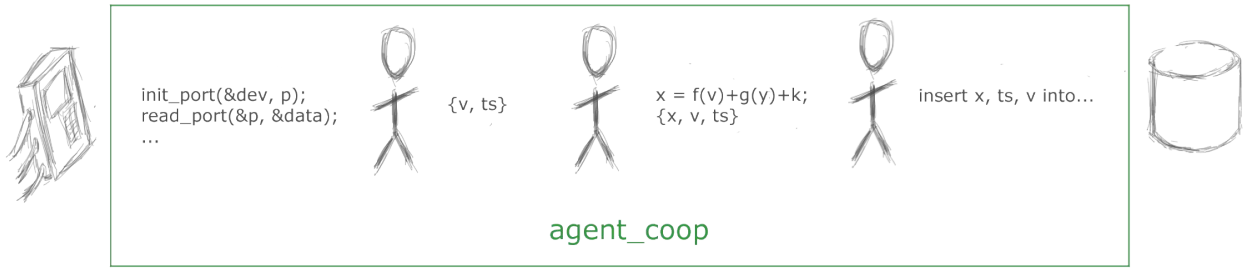
Coops of agents

A group of agents have to work together.



Coops of agents

That group is a coop of agents



Making a coop

```
so_5::environment_t & env = ...;
auto coop = env.make_coop(); // Make an instance of new coop.

// Fill the coop. Bind agents to the appropriate dispatchers.

coop->make_agent_with_binder<device_reader>(
    so_5::disp::one_thread::make_dispatcher(env, "device").binder(), ... );

coop->make_agent<data_processor>(...);

coop->make_agent_with_binder<db_writer>(
    so_5::disp::one_thread::make_dispatcher(env, "db").binder(), ... );

// Register the coop.
env.register_coop(std::move(coop));
```

Making a coop

```
so_5::environment_t & env = ...;
auto coop = env.make_coop(); // Make an instance of new coop.

// Fill the coop. Bind agents to the appropriate dispatchers.

coop->make_agent_with_binder<device_reader>(
    so_5::disp::one_thread::make_dispatcher(env, "device").binder(), ... );

coop->make_agent<data_processor>(...);

coop->make_agent_with_binder<db_writer>(
    so_5::disp::one_thread::make_dispatcher(env, "db").binder(), ... );

// Register the coop.
env.register_coop(std::move(coop));
```

Making a coop

```
so_5::environment_t & env = ...;
auto coop = env.make_coop(); // Make an instance of new coop.

// Fill the coop. Bind agents to the appropriate dispatchers.

coop->make_agent_with_binder<device_reader>(
    so_5::disp::one_thread::make_dispatcher(env, "device").binder(), ... );

coop->make_agent<data_processor>(...);

coop->make_agent_with_binder<db_writer>(
    so_5::disp::one_thread::make_dispatcher(env, "db").binder(), ... );

// Register the coop.
env.register_coop(std::move(coop));
```

Making a coop

```
so_5::environment_t & env = ...;
auto coop = env.make_coop(); // Make an instance of new coop.

// Fill the coop. Bind agents to the appropriate dispatchers.

coop->make_agent_with_binder<device_reader>(
    so_5::disp::one_thread::make_dispatcher(env, "device").binder(), ... );

coop->make_agent<data_processor>(...);

coop->make_agent_with_binder<db_writer>(
    so_5::disp::one_thread::make_dispatcher(env, "db").binder(), ... );

// Register the coop.
env.register_coop(std::move(coop));
```

Making a coop (simplified version)

```
so_5::environment_t & env = ...;

// Ask SObjectizer to make a new coop and then register it.
env.introduce_coop([&](so_5::coop_t & coop) {
  // Fill the coop. Bind agents to the appropriate dispatchers.
  coop.make_agent_with_binder<device_reader>(
    so_5::disp::one_thread::make_dispatcher(env, "device").binder(),  ... );

  coop.make_agent<data_processor>(...);

  coop.make_agent_with_binder<db_writer>(
    so_5::disp::one_thread::make_dispatcher(env, "db").binder(), ... );

}); // Coop will be registered automatically.
```

Making a coop (simplified version)

```
so_5::environment_t & env = ...;

// Ask SObjectizer to make a new coop and then register it.
env.introduce_coop([&](so_5::coop_t & coop) {
  // Fill the coop. Bind agents to the appropriate dispatchers.
  coop.make_agent_with_binder<device_reader>(
    so_5::disp::one_thread::make_dispatcher(env, "device").binder(),  ... );

  coop.make_agent<data_processor>(...);

  coop.make_agent_with_binder<db_writer>(
    so_5::disp::one_thread::make_dispatcher(env, "db").binder(), ... );

}); // Coop will be registered automatically.
```

Making a coop (simplified version)

```
so_5::environment_t & env = ...;

// Ask SObjectizer to make a new coop and then register it.
env.introduce_coop([&](so_5::coop_t & coop) {
  // Fill the coop. Bind agents to the appropriate dispatchers.
  coop.make_agent_with_binder<device_reader>(
    so_5::disp::one_thread::make_dispatcher(env, "device").binder(),  ... );

  coop.make_agent<data_processor>(...);

  coop.make_agent_with_binder<db_writer>(
    so_5::disp::one_thread::make_dispatcher(env, "db").binder(), ... );

}); // Coop will be registered automatically.
```

Agent-based ping-pong example (1)

```
#include <so_5/all.hpp>

struct ping final : public so_5::signal_t {};
struct pong final : public so_5::signal_t {};
```

Agent-based ping-pong example (2)

```
class pinger final : public so_5::agent_t {
public:
  pinger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_start() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pongs_{};

  void on_pong(mhood_t<pong>);
};
```

Agent-based ping-pong example (2)

```
class pinger final : public so_5::agent_t {
public:
  pinger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_start() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pongs_{};

  void on_pong(mhood_t<pong>);
};
```

Agent-based ping-pong example (2)

```
class pinger final : public so_5::agent_t {
public:
  pinger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_start() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pongs_{};

  void on_pong(mhood_t<pong>);
};
```

Agent-based ping-pong example (2)

```
class pinger final : public so_5::agent_t {
public:
  pinger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_start() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pongs_{};

  void on_pong(mhood_t<pong>);
};
```

Agent-based ping-pong example (2)

```
class pinger final : public so_5::agent_t {
public:
  pinger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_start() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pongs_{};

  void on_pong(mhood_t<pong>);
};
```

Agent-based ping-pong example (2)

```
class pinger final : public so_5::agent_t {
public:
  pinger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_start() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pongs_{};

  void on_pong(mhood_t<pong>);
};
```

Agent-based ping-pong example (2)

```
class pinger final : public so_5::agent_t {
public:
  pinger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_start() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pongs_{};

  void on_pong(mhood_t<pong>);
};
```

Agent-based ping-pong example (3)

```
void pinger::so_define_agent() {
  so_subscribe(mbox_).event(&pinger::on_pong);
}

void pinger::so_evt_start() {
  so_5::send<ping>(mbox_);
}

void pinger::so_evt_finish() {
  std::cout << "pongs received: " + std::to_string(pongs_) << std::endl;
}

void pinger::on_pong(mhood_t<pong>) {
  ++pongs_;
  so_5::send<ping>(mbox_);
}
```

Agent-based ping-pong example (3)

```
void pinger::so_define_agent() {
  so_subscribe(mbox_).event(&pinger::on_pong);
}

void pinger::so_evt_start() {
  so_5::send<ping>(mbox_);
}

void pinger::so_evt_finish() {
  std::cout << "pongs received: " + std::to_string(pongs_) << std::endl;
}

void pinger::on_pong(mhood_t<pong>) {
  ++pongs_;
  so_5::send<ping>(mbox_);
}
```

Agent-based ping-pong example (3)

```
void pinger::so_define_agent() {
  so_subscribe(mbox_).event(&pinger::on_pong);
}

void pinger::so_evt_start() {
  so_5::send<ping>(mbox_);
}

void pinger::so_evt_finish() {
  std::cout << "pongs received: " + std::to_string(pongs_) << std::endl;
}

void pinger::on_pong(mhood_t<pong>) {
  ++pongs_;
  so_5::send<ping>(mbox_);
}
```

Agent-based ping-pong example (3)

```
void pinger::so_define_agent() {
  so_subscribe(mbox_).event(&pinger::on_pong);
}

void pinger::so_evt_start() {
  so_5::send<ping>(mbox_);
}

void pinger::so_evt_finish() {
  std::cout << "pongs received: " + std::to_string(pongs_) << std::endl;
}

void pinger::on_pong(mhood_t<pong>) {
  ++pongs_;
  so_5::send<ping>(mbox_);
}
```

Agent-based ping-pong example (3)

```
void pinger::so_define_agent() {
  so_subscribe(mbox_).event(&pinger::on_pong);
}

void pinger::so_evt_start() {
  so_5::send<ping>(mbox_);
}

void pinger::so_evt_finish() {
  std::cout << "pongs received: " + std::to_string(pongs_) << std::endl;
}

void pinger::on_pong(mhood_t<pong>) {
  ++pongs_;
  so_5::send<ping>(mbox_);
}
```

Agent-based ping-pong example (4)

```
class ponger final : public so_5::agent_t {
public:
  ponger(context_t ctx, so_5::mbox_t mbox) : so_5::agent_t{std::move(ctx)}, mbox_{std::move(mbox)} {}

  void so_define_agent() override;

  void so_evt_finish() override;

private:
  const so_5::mbox_t mbox_;
  unsigned long long pings_{};

  void on_ping(mhood_t<ping>);
};
```

Agent-based ping-pong example (5)

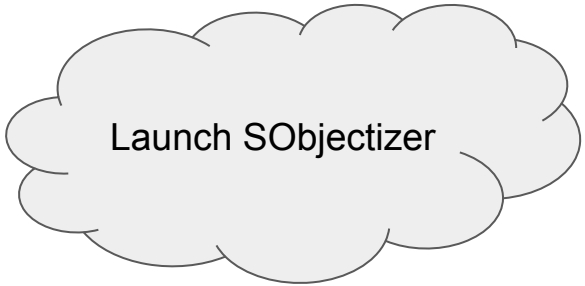
```
void ponger::so_define_agent() {
  so_subscribe(mbox_).event(&ponger::on_ping);
}

void ponger::so_evt_finish() {
  std::cout << "pings received: " + std::to_string(pings_) << std::endl;
}

void ponger::on_ping(mhood_t<ping>) {
  ++pings_;
  so_5::send<pong>(mbox_);
}
```

Agent-based ping-pong example (6)

```
int main() {
  so_5::launch([](so_5::environment_t & env) {
    env.introduce_coop(
        so_5::disp::active_obj::make_dispatcher(env).binder(),
        [&](so_5::coop_t & coop) {
          const auto mbox = env.create_mbox();

          coop.make_agent<pinger>(mbox);
          coop.make_agent<ponger>(mbox);
        } );

    std::this_thread::sleep_for(std::chrono::seconds{1});
    env.stop();
  });
}
```

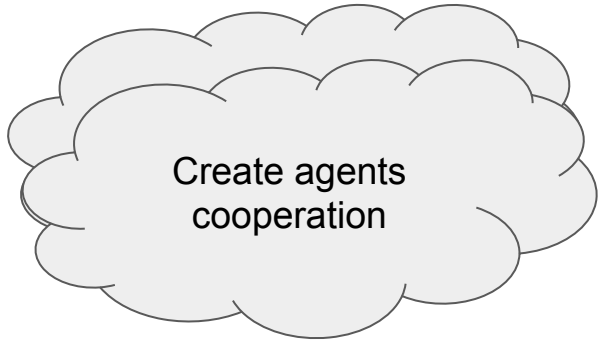
Agent-based ping-pong example (6)

```
int main() {
  so_5::launch([](so_5::environment_t & env) {
    env.introduce_coop(
        so_5::disp::active_obj::make_dispatcher(env).binder(),
        [&](so_5::coop_t & coop) {
          const auto mbox = env.create_mbox();

          coop.make_agent<pinger>(mbox);
          coop.make_agent<ponger>(mbox);
        } );

    std::this_thread::sleep_for(std::chrono::seconds{1});
    env.stop();
  });
}
```



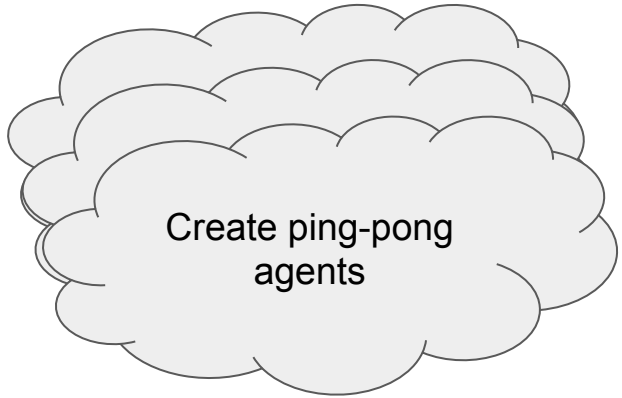
Agent-based ping-pong example (6)

```
int main() {
  so_5::launch([](so_5::environment_t & env) {
    env.introduce_coop(
        so_5::disp::active_obj::make_dispatcher(env).binder(),
        [&](so_5::coop_t & coop) {
          const auto mbox = env.create_mbox();

          coop.make_agent<pinger>(mbox);
          coop.make_agent<ponger>(mbox);
        } );

    std::this_thread::sleep_for(std::chrono::seconds{1});
    env.stop();
  });
}
```



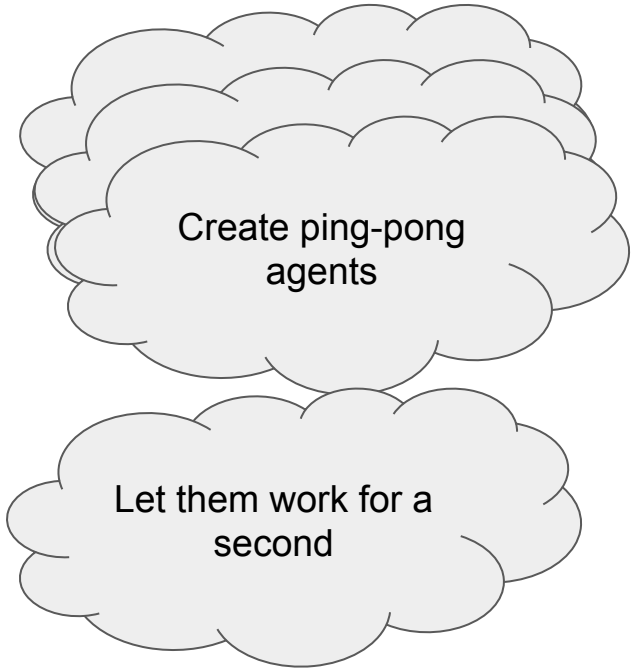
Agent-based ping-pong example (6)

```
int main() {
  so_5::launch([](so_5::environment_t & env) {
    env.introduce_coop(
        so_5::disp::active_obj::make_dispatcher(env).binder(),
        [&](so_5::coop_t & coop) {
          const auto mbox = env.create_mbox();

          coop.make_agent<pinger>(mbox);
          coop.make_agent<ponger>(mbox);
        } );

    std::this_thread::sleep_for(std::chrono::seconds{1});
    env.stop();
  });
}
```



Agent-based ping-pong example (6)

```
int main() {
  so_5::launch([](so_5::environment_t & env) {
    env.introduce_coop(
        so_5::disp::active_obj::make_dispatcher(env).binder(),
        [&](so_5::coop_t & coop) {
          const auto mbox = env.create_mbox();

          coop.make_agent<pinger>(mbox);
          coop.make_agent<ponger>(mbox);
        } );

    std::this_thread::sleep_for(std::chrono::seconds{1});
    env.stop();
  });
}
```



CSP-channels

Let's speak about another style: raw threads and mchains

What is mchain?

Mchain (message chain) is a message queue.

Mchain can contain messages of different types at the same time.

Messages are sent by usual send() functions.

Messages are extracted by receive() and select() functions.

- Several threads can call receive()/select() for a mchain at the same time.
- Only one thread will get a message.

Two kinds of mchains

Size-unlimited mchain

- grows dynamically;
- can contain as many messages as allows available memory;
- never blocks on send().

Size-limited mchain

- max capacity specified at the construction time;
- can block send() operation;

Size-limited mchains are good when overload-control is necessary.

Mchain-based ping-pong example (1)

```
#include <so_5/all.hpp>

struct ping {
  int counter_;
};

struct pong {
  int counter_;
};
```

Mchain-based ping-pong example (2)

```
void pinger_proc(so_5::mchain_t self_ch, so_5::mchain_t ping_ch) {
  so_5::send<ping>(ping_ch, 1000); // The initial "ping".

  // Read all message until channel will be closed.
  so_5::receive(so_5::from(self_ch).handle_all(),
    [&](so_5::mhood_t<pong> cmd) {
      if(cmd->counter_ > 0)
        so_5::send<ping>(ping_ch, cmd->counter_ - 1);
      else {
        // Channels have to be closed to break `receive` calls.
        so_5::close_drop_content(self_ch);
        so_5::close_drop_content(ping_ch);
        }
    });
}
```

Mchain-based ping-pong example (2)

```
void pinger_proc(so_5::mchain_t self_ch, so_5::mchain_t ping_ch) {
  so_5::send<ping>(ping_ch, 1000); // The initial "ping".

  // Read all message until channel will be closed.
  so_5::receive(so_5::from(self_ch).handle_all(),
    [&](so_5::mhood_t<pong> cmd) {
      if(cmd->counter_ > 0)
        so_5::send<ping>(ping_ch, cmd->counter_ - 1);
      else {
        // Channels have to be closed to break `receive` calls.
        so_5::close_drop_content(self_ch);
        so_5::close_drop_content(ping_ch);
        }
    });
}
```

Mchain-based ping-pong example (2)

```
void pinger_proc(so_5::mchain_t self_ch, so_5::mchain_t ping_ch) {
  so_5::send<ping>(ping_ch, 1000); // The initial "ping".

  // Read all message until channel will be closed.
  so_5::receive(so_5::from(self_ch).handle_all(),
    [&](so_5::mhood_t<pong> cmd) {
      if(cmd->counter_ > 0)
        so_5::send<ping>(ping_ch, cmd->counter_ - 1);
      else {
        // Channels have to be closed to break `receive` calls.
        so_5::close_drop_content(self_ch);
        so_5::close_drop_content(ping_ch);
        }
    });
}
```

Mchain-based ping-pong example (2)

```
void pinger_proc(so_5::mchain_t self_ch, so_5::mchain_t ping_ch) {
  so_5::send<ping>(ping_ch, 1000); // The initial "ping".

  // Read all message until channel will be closed.
  so_5::receive(so_5::from(self_ch).handle_all(),
    [&](so_5::mhood_t<pong> cmd) {
      if(cmd->counter_ > 0)
        so_5::send<ping>(ping_ch, cmd->counter_ - 1);
      else {
        // Channels have to be closed to break `receive` calls.
        so_5::close_drop_content(self_ch);
        so_5::close_drop_content(ping_ch);
        }
    });
}
```

Mchain-based ping-pong example (3)

```
void ponger_proc(so_5::mchain_t self_ch, so_5::mchain_t pong_ch) {
  int pings_received{};

  // Read all message until channel will be closed.
  so_5::receive(so_5::from(self_ch).handle_all(),
    [&](so_5::mhood_t<ping> cmd) {
        ++pings_received;
        so_5::send<pong>(pong_ch, cmd->counter_);
    });

  std::cout << "pings received: " << pings_received << std::endl;
}
```

Mchain-based ping-pong example (3)

```
void ponger_proc(so_5::mchain_t self_ch, so_5::mchain_t pong_ch) {
  int pings_received{};

  // Read all message until channel will be closed.
  so_5::receive(so_5::from(self_ch).handle_all(),
    [&](so_5::mhood_t<ping> cmd) {
        ++pings_received;
        so_5::send<pong>(pong_ch, cmd->counter_);
    });

  std::cout << "pings received: " << pings_received << std::endl;
}
```

Mchain-based ping-pong example (4)

```
int main() {
  so_5::wrapped_env_t sobj;

  auto pinger_ch = so_5::create_mchain(sobj);
  auto ponger_ch = so_5::create_mchain(sobj);

  std::thread pinger{pinger_proc, pinger_ch, ponger_ch};
  std::thread ponger{ponger_proc, ponger_ch, pinger_ch};

  ponger.join();
  pinger.join();

  return 0;
}
```

Mchain-based ping-pong example (4)

```
int main() {
  so_5::wrapped_env_t sobj;

  auto pinger_ch = so_5::create_mchain(sobj);
  auto ponger_ch = so_5::create_mchain(sobj);

  std::thread pinger{pinger_proc, pinger_ch, ponger_ch};
  std::thread ponger{ponger_proc, ponger_ch, pinger_ch};

  ponger.join();
  pinger.join();

  return 0;
}
```

Mchain-based ping-pong example (4)

```
int main() {
  so_5::wrapped_env_t sobj;

  auto pinger_ch = so_5::create_mchain(sobj);
  auto ponger_ch = so_5::create_mchain(sobj);

  std::thread pinger{pinger_proc, pinger_ch, ponger_ch};
  std::thread ponger{ponger_proc, ponger_ch, pinger_ch};

  ponger.join();
  pinger.join();

  return 0;
}
```

Epilog

SObjectizer simplifies development

SObjectizer is in use for more than 18 years.

It's a good tool that proved its efficiency for us. Many times.

Former students start write correct multithreaded code just after a short time of studying SObjectizer.

What distinguishes SObjectizer?

Mature and stable

Cross-platform: works on Windows, Linux, FreeBSD, macOS, and Android.

Easy-to-use: some users studied SObjectizer even without asking for our help.

Free: distributed under BSD-3-CLAUSE license.

It's not dead.

It evolves

Some of features added since 2013:

- agents as hierarchical state machines;
- mutability for messages;
- message chains;
- environment infrastructures;
- enveloped messages;
- dead-letter handlers;
- message-tracing;
- stop-guards;
- run-time monitoring;
- unit-testing of agents;
- ...

so5extra

Some additional goodies not included into SObjectizer's core:

- dispatchers on top of Asio library;
- additional types of mboxes;
- tools for synchronous interactions;
- and some more...

Live in a separate repo: https://github.com/Stiffstream/so5extra

That's all. Thanks you!

SObjectizer: https://github.com/Stiffstream/sobjectizer
Docs: https://github.com/Stiffstream/sobjectizer/wiki

so5extra: https://github.com/Stiffstream/so5extra
Docs: https://github.com/Stiffstream/so5extra/wiki

Support if you need it: https://stiffstream.com